R and Data Analysis

fwang2@ornl.gov

Contents

1 Data

1.1 Vector

Vector is an ordered list of elements of same mode. Among the mode R has: 1) logical 2) character 3) numeric 4) complex 5) raw

In R, a vector is an object with order and length, but no dimensions. A matrix in R however, has dimension. This is different from Matlab, where vector is just a special matrix. If you run `dim()`, `nrow()`, `ncol()` on a vector object, `NULL` is returned.

```
x = vector()  # vector of length 0
y = c("alice", "in", "wonderland")
z = paste(y[1], y[2], sep="-") # default sep is space
```

Now you can convert a vector into a column vector (or matrix in R) by calling `as.matrix()` on the vector object.

Collect vectors into a data frame:

```
 > Year = c(1800, 1900)
 > Carbon = c(8,1630)
 > fossilefuel = data.frame(year=Year, carbon=Carbon)
 > fossilefuel
   year        carbon
1   1800        8
2   1900        1630
```

Subset of vectors

You can specific the indices of the elements are to be extracted, use negative script to omit the elements in the nomiated subscript poitions:

```
 > x = c(3, 8, 2, 5, 100, 70)
 > x[c(2,4)]
[1] 8 5
> x[-c(2,4)]
[1]   3   2 100  70
> x>10
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```
> x[x>10]
[1] 100  70
```

1.2 Data Frame

A data frame is a collection of variables which share many of the properties of matrices and lists. More precisely, a data frame is a list of variables of the same length with unique row names. It is given class name "data.frame".

Noted that when you use `scan()` to read data file, the return object is a vector; but if you use `read.table()`, then the return object is a data frame, which has dimensions.

Get Subset from Data Frame

- `salary$id` is one way to access a variable inside the data frame
- `salary$id[1]` - access first element of the vector
- `alary$id[2:3]` - access a subset of elements
- `salary$id[c(5,7)]` - access elements 5 and 7
- `salary$id[ salary$[salary$startyr == 95]` - all people join in 95
- `salary[1,]` - get the first row
- `salary[,2]` - get the second column
- `salary[3:7, 2:4]` - some rows and columns
- `salary[, c("id","temp")]` - extract columns by name
- `yes = antibiotics[antibiotics$antib == 1, ]` - put this subset into a new data frame

1.2.1 Slice Data Set

DDN dataset has a column named "Operation", with categorical value of "read" and "write". Suppose I want all rows of write operation go into a data frame, here is how to do that:

```
ddn = read.csv("ddnraw.csv")
ind = ddn$Operation == "write"
ddnwrite = ddn[ind,]
```

Obviously, you can also slice data horizontally based on some kind of row based criteria.

1.3 Matrix

Matrix: two or more vectors of the same data type and same length.

```
y = cbind(letters[1:4],LETTERS[1:4])
```

`cbind()` binds vectors as columns.

1.4 Factors

Factors represent categorical variables. You can't do math operations on them (except for ==). An example of such is sex column which has two values: male and female. You can table such data by `table(students$sex)` and R will do frequency distribution on them.

1.5 Sequences of data

- `seq()` is more general than `1:5`:

```
>seq(from=5, to=22, by=3)
```

- To repeat sequence (4,5) three times:

```
> rep(c(4,5), 3)
```

1.6 Selection and Matching

You can select data by providing a matching check:

```
> student[score > 80]
```

You can also use if function to test the truth:

```
> distance = c(148,182,173,166,109,414,166)
> dist.sort = if (distance[1] > 150)
+ sort(distance, decreasing=T) else sort(distance)
> dist.sort
[1] 109 148 166 166 173 182 414
```

1.7 Examine Data Type

You can use `mode()` to check the data type. For example:

```
> mode(mean)
[1] "function"
> mode(1)
[1] "numeric"
> mode(c(TRUE,FALSE))
[1] "logical"
```

`typeof()` determines the (R internal) type or storage mode for given object. For example:

```
> typeof(2)
[1] "double"
> mode(2)
[1] "numeric"
```

Setting a mode from one to anther is called **coercion**. `is.x()` and `as.x()` can be used to check and set the mode. `x` here can be "matrix", "vector", "list", "numeric", "logical", "integer" etc.

You can access and change an object's attributes by `attributes()`. For object such as array, matrix, or data frame, you can use `dim()` to get its dimension.

1.8 Read Directory

`dir()` can return all files under current directory. A few variations on this function: it can take both a path name and regular expression to select what files to return. A convenient function `glob2rx()` converts wildcard expression as seen in filesystem to a regular expression:

```
> dir(".", glob2rx("jj*.dat"))
[1] "jj.dat"  "jj2.dat"
```

1.9 Read data

Before you read files, know what directory you start with: `getwd()` and `setwd()` show and change current work directory respectively.

Read raw data

```
d = scan(file="raw.dat")
```

Read text Data

if data salary.txt is like this:

```
id gender startyr
1   F       95
2   F       93
3   M       97
```

You can read the data as:

```
salary = read.table("salary.txt", header=TRUE)
```

if variable name are not included, you can supply them:

```
salary=read.table("salary.txt", col.names=c("id","gender","startyr"))
```

if data set is like this:

```
Ozone, Solar, Wind, Tmp
41, 190, 68, 5
NA, NA, 14, 5
```

Then we have at least two ways to read the data:

```
ozone = read.table("ozone.csv", header=TRUE, sep=",")
ozone = read.csv("ozone.csv")
```

You can also change column names after reading the data:

```
> d = read.table('A.txt')
> names(d)
[1] "V1" "V2"
```

```
> names(d) = 1:length(d)             --> column name 1 2
> names(d) = LETTERS[1:length(d)]    --> column name A B
```

1.10 Read Performance

If your file only contains number, or only strings, it is wiser to store it in a matrix, not a data.frame - that is what function `scan()` does.

For large files, you can assiste R to read faster by telling R the type of each columns.

```
# the first column is numeric, the other contain strings
read.table("foo.txt", colClasses = c("numeric", rep("character", 10)))
```

1.11 Create Data

Sometimes, the program generates bunch of data that you want to save it for later analysis. An example of doing this is:

```
# 100x8 empty matrix
outp = matrix(nrow=100, ncol=2)
for (i in 1:100) {
    ...
    outp[i,] = c(data1, data2)
}
write.table(outp, file="outp.Rdata", row.names=FALSE,
    col.names=c("kurt1", "kurt2"))
```

1.12 Write Data

There should be better ways, but here is what I know: say I want to write a matrix with specific rows and columns on disk:

```
x = matrix(nrow=10,ncol=3)
x[1,] = c (1,2,3)
...
write.table(x,file="data.txt",row.names=FALSE,col.names=c("min","max","sd"))
```

You can also save R objects in a binary file and load it into workspace as it is:

```
x = runif(20)
y = list(a=1,b=TRUE,c="oops")
save(x,y,file="xy.Rdata)
```

2 Functions

2.1 Configuration

- `options(digits=3)` - ask R to print no more than 3 decimal digits.
- A simple way of trimming is to call `round(num, digits=3)`.

2.2 Statistic

- `names()` access variable names in a data frame.
- `mean()`, `median()`, `var()`, and `sd()` is self-obvious.
- `runif(5,0,2)`: return 5 random values from 0 to 2 with equal probability

3 Programming R

You load a program source by `source()`. You can also call invoke R from command line by:

```
Rscript some.R arg1 arg2
```

In R, a *library* is a directory, usually have one or more packages. A *package* is a set of functions, data sets and manual pages, contained in a directory, or a *.tar.gz file. A *bundle* is a set of packages contained in the same *.tar.gz files. The confusing part is, to load a package, your call `library()`.

There are some steps to follow if you want your own R script to be a package, refer to `http://zoonek2.free.fr/UNIX/48_R/02.html` for details.

3.1 Control Structure

Conditional

```
if (...) {
    ...
} else {
    ...
}

x = if (...) 3.14 else 2.71
```

Here is an interesting way of constructing vector from conditional expression:

```
x = rnorm(100)
y = ifelse( x > 0, 1, -1)
z = ifelse( x > 0, 1, ifelse( x<0, -1, 0))
```

Loop

```
for (i in 1:10) {
    ...
    if (...) {next}

    if (...) {break}
    ...
}
##
while (...) {
    ...
}
##
repeat {
    ...
```

```
    if (...) { break }
    ...
}
```

3.1.1 Example

```
x = as.logical(as.integer(runif(5,0,2)))
# coerced into logical values
y = vector();
for (i in 1:length(x)) {
    if (x[i]) {y[i] = 1} else {y[i]=0}
```

Another example for avoiding loops:

```
ifelse(x, y = 1, y = 0)
# this function ifelse(a,b,c) executes, element by element, b[i]
# if a[i] is TRUE, c[i] if a[i] is FALSE.
```

3.2 Global variable

Assign an object to be globally available:

```
a.function = function(z) {
    y <<- 2 *z
    y
}
```

3.3 Command Line

```
require("moments")

argv = commandArgs(TRUE)
file = argv[1]
d = scan(file=paste(file))
d_min = min(d)
d_max = max(d)
d_norm = (d-d_min)/d_min
kurt = kurtosis(d_norm)
cat("\tKurtosis = ", kurt, "\n")
```

4 R Environment and Customization

- `objects()` - show all objects in current workspace.
- `sessionInfo()` - show which packages are currently attached
- `data()` - get a list of dataset in all packages

On Mac, the customization file at `$HOME/.Rprofile`. `.Rdata` is saved in the current directory.

5 Graphing

A common question on this is to add grid line, which is really easy: `grid(col="darkgray")` for example.

5.1 Customize Graphic Parameters

You can customize these parameters in each plot, you can also do so with `par()` function. By itself, it displays all settings you can customize.

```
par()
opar = par()
par(col.lab="red") # red x and y labels
hist(mtcar$mpg)
par(opar)          # restore original setting
```

Text and symbol size is controlled by:

```
cex         overall text and symbol size, 1 is default
cex.axis    magnification of axis annotation related to cex
cex.lab     x,y label size related to cex
cex.main    title size related to cex
cex.sub     subtitle size related to cex
```

Plot symbol and line style

Plot symbol is controlled by `pch=` options, from 0 (square), 1 (circle), 2 (triangle), 3 (cross), all the way to 25. For line, line type is controlled by `lty=` option, from 1 (solid), 2 (dashed), 3 (dotted) etc; line width is controlled by `lwd=`.

Colors controls include:

```
col         default plotting color, may take vector of colors to recycle
col.axis    color for axis annotation
col.lab     color for x, y labels
col.main    color for title
col.sub     color for subtitles
fg          plot foreground colors
bg          plot for background colors
```

You can create a vector of *n* contiguous colors using function of `rainbow(n)`, `heat.colors(n)`, `terrain.colors(n)`, `topo.colors(n)`, and `cm.colors(n)`.

```
theta = 1:50
plot(theta,sin(theta), col=1:50, pch=16, cex=4)
plot(theta,cos(theta), col=colors()[51:100], pch=16, cex=4)
\end{Verbatm}


\rhead{Font family control (On Windows):}

\begin{Verbatim}
> plot(1:10)
> windowsFonts(A=windowsFont("Cambria"))
> text(5,8, family="A", "hello world")
```

Font style control:

```
font        1=plain,2=bold,3=italic,4=bold italic,5=symbol
font.axis   font for axis annotation
font.lab    font for x, y labels
font.main   font for title
font.sub    font for subtitles
ps          font point size, text size = ps*cex
family      font family for drawing text: serif, sans, mono, symbol
```

You can see what font is mapped to which family by run `windowsFonts()`.

Margin and graph size

```
## on windows
windows(height=4, width=6)

## on unix
x11(height=4, width=6)
```

Margin is controlled by:

```
mar         numerical vector indicating margin size
            c(bottom, left, top, right) in lines
            default = c(5,4,4,2) + 0.1
mai         margin size in inches
pin         plot dimension (width, height) in inches
```

5.2 Line Plot

```
plot(ge, ylab="Earnings per Share", main="General Electronics")
```

- `type="o"` - draw data point in circles
- `col="blue"` - draw line in color blue
- `lty="dashed"` - dashed line
- `lowess()` and `supsmu()` are scatterplot smoothers. They draw smooth curves that fit the relationship between y and x.

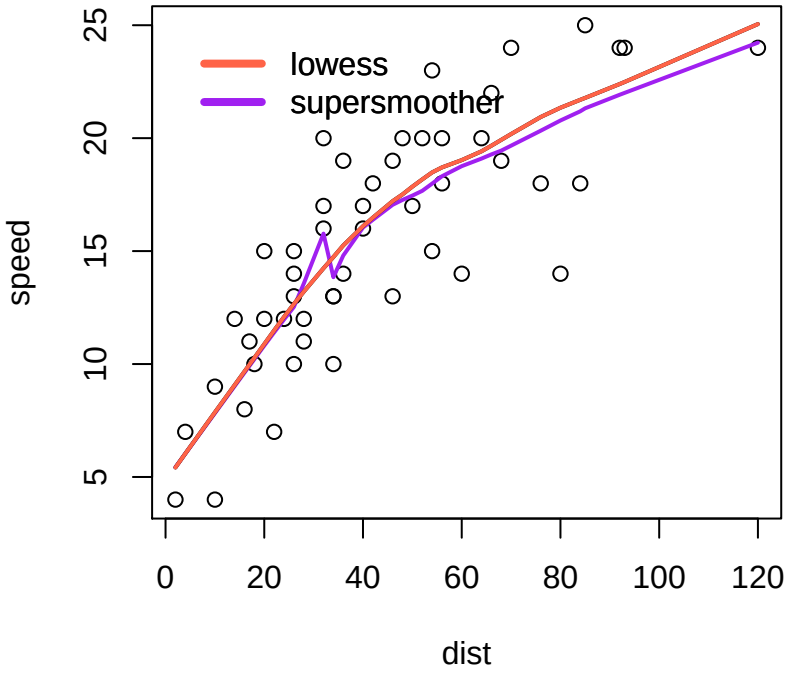
Example

```
> data(cars)
> plot(speed~dist,data=cars)
> with(cars,lines(supsmu(dist,speed),col="purple",lwd=2))
> with(cars,lines(lowess(dist,speed),col="tomato",lwd=2))
> legend(2,25, legend=c("lowess","supersmoother"),bty="n", lwd=4, col=c("tomato", "purple"))
```

5.3 Bar Plot

```
# Grouped Bar Plot
counts <- table(mtcars$vs, mtcars$gear)
barplot(counts, main="Car Distribution by Gears and VS",
          legend =c("d1","d 2"),  col=c("gray","greenyellow"),
          beside=TRUE, axis.lty=1,
          names.arg=c("3 Gears", "4 Gears", "5   Gears"), cex.names=0.8)
```

`axis.lty=1` is to make axis show up - in the default mode, this axis doesn't get displayed.



5.4 Ploting Two Data Series on One Figure

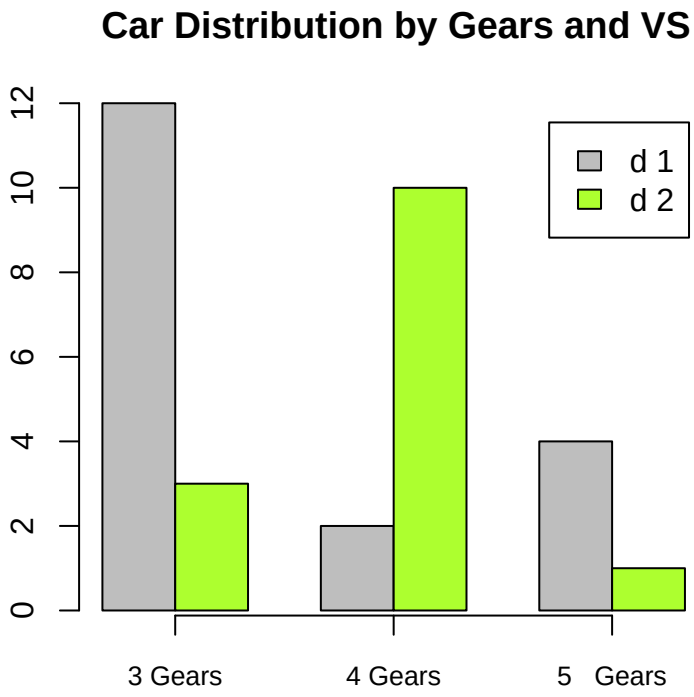
This is often achieved with `hold` commmand on Matlab, but a bit less intutive on R:

```
# plot overlapping ones
# assuming d2 and d3 are holding two data series
pdf(file="chester-compare.pdf", family="Times")
ymax = max(d2, d3)
ymin = min(d2, d3)
plot(d2, type="l", col="red", ylim=c(ymin, ymax), log="y",
    main="Chester Stock Kernel vs. Low Noise Kernel (FWQ)",
    xlab="Samples",
    ylab="Cycle Counts (Log Scaled)")
lines(d3, type="l", col="blue")
legend(13000, ymax, c("2.2 Stock Kernel", "2.2 Low Noise Kernel"), cex=0.8,
    fill = c("red", "blue"))
```

Noted that since we plot d3 later, the graphic lines from d3 will overwrite d2. This essentially give d3 a higher visibility.

5.5 Export to PDF and PNG

To generate pdf, you can do:




```
pdf("data.pdf",height=4,width=6, family="Times")

dev.off()     # must close
```

You can issue many pdf commands in between before you close the device file. The family option is quite limited. What works for png doesn't ncessarily work for pdf.

Here are convenient functions that are designed to generate both png and pdf in one shot:

```
openg <- function(width=3, height=3, pointsize=8)
{
    windows(width=width, height=height, pointsize = pointsize)
}

# above windows is 3x3 inch

saveg <- function (fn, width=3, height=3, pointsize=8)
{
    dev.copy(device= pdf, file=paste(fn, ".pdf", sep=""),
        width=width, height = height, pointsize=pointsize)

    dev.off()

    dev.copy(device= png, file=paste(fn, ".png", sep=""),
        width=width, height = height, pointsize = pointsize)

    dev.off()
 }


 # save figure to both pdf and png format
```

```
 # save('a-plot')
 # save('b-plot', 5, 4, 11)

h <- function(x, xlab= '', ylab = 'density', main = '', ...)
{
    hist(x, xlab=xlab, main='', freq=FALSE, col = 'grey90', ylab=ylab,
    ...)

}
```

6 Probability

6.1 Generate values from a distribution

binomial

```
> rbinom(10, size=1, p=0.5)
 [1] 1 1 0 1 0 1 1 1 0 1
```

Poisson

```
rpois(20, 3)
```

Normal

```
rnorm(10)
```

Uniform

```
runif(n=20, min=0, max=1)
```

Exponential

```
rexp(n=10, rate=3)
```